```
US-PAT-NO:              6438664

DOCUMENT-IDENTIFIER:    US 6438664 B1

TITLE:                  Microcode patch device and method for
patching microcode
                       using match registers and patch
routines

DATE-ISSUED:            August 20, 2002

INVENTOR-INFORMATION:
NAME                               CITY
STATE     ZIP CODE  COUNTRY
McGrath; Kevin J.                  Los Gatos                CA
     N/A       N/A
Pickett; James K.                  Austin                   TX
     N/A       N/A

ASSIGNEE INFORMATION:
NAME                         CITY                        STATE
ZIP CODE  COUNTRY   TYPE CODE
Advanced Micro Devices,      Sunnyvale                   CA
N/A       N/A       02
Inc.


APPL-NO:        09/ 428635

DATE FILED:     October 27, 1999


INT-CL:              [07] G06F009/00,G06F012/00 ,G06F013/00

US-CL-ISSUED:        711/154, 711/102 , 711/165 , 711/202 ,
712/245

US-CL-CURRENT:       711/154, 711/102 , 711/165 , 711/202 ,
712/245


FIELD-OF-SEARCH:     711/102; 711/103 ; 711/202 ; 711/213 ;
711/154 ; 711/165
                     ; 712/244 ; 712/245 ; 714/8
```

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME
	US-CL	
4028678	June 1977	Moran
N/A	**N/A**	N/A
4028679	June 1977	**Divine**
N/A	N/A	N/A
4028683	June 1977	Divine et al.
N/A	**N/A**	N/A
4028684	June 1977	**Divine** et al.
N/A	N/A	N/A
4044338	August 1977	Wolf
N/A	**N/A**	N/A
4319079	March 1982	**Best**
N/A	N/A	N/A
4400798	August 1983	Francis et al.
N/A	**N/A**	N/A
4453212	June 1984	**Gaither** et al.
N/A	N/A	N/A
4482953	November 1984	Burke
N/A	**N/A**	N/A
4542453	September 1985	**Patrick** et al
N/A	N/A	N/A
4577319	March 1986	Takeuchi et al.
N/A	**N/A**	N/A
4610000	September 1986	**Lee**
N/A	N/A	N/A
4751703	June 1988	Picon et al.
N/A	**N/A**	N/A
4802119	January 1989	**Heene** et al.
N/A	N/A	N/A
4807115	February 1989	Torng
N/A	**N/A**	N/A
4807857	February 1989	**Wolf** et al.
N/A	N/A	N/A
4839797	June 1989	Katori et al.
N/A	**N/A**	N/A
4857612	August 1989	**Bacskai**
N/A	N/A	N/A
4858105	August 1989	Kuriyama et al.
N/A	**N/A**	N/A
4928223	May 1990	**Dao** et al.
N/A	N/A	N/A
4982360	January 1991	Johnson et al.

N/A	**N/A**	N/A
5053631	October 1991	**Perlman** et al.
N/A	N/A	N/A
5053949	October 1991	Allison et al.
N/A	**N/A**	N/A
5058048	October 1991	**Gupta** et al.
N/A	N/A	N/A
5115500	May 1992	Larsen
N/A	**N/A**	N/A
5129067	July 1992	**Johnson**
N/A	N/A	N/A
5136697	August 1992	Johnson
N/A	**N/A**	N/A
5167026	November 1992	**Murray** et al.
N/A	N/A	N/A
5202967	April 1993	Matsuzaki et al.
N/A	**N/A**	N/A
5212693	May 1993	**Chao** et al.
N/A	N/A	N/A
5226126	July 1993	McGarland et al.
N/A	**N/A**	N/A
5226130	July 1993	**Favor** et al.
N/A	N/A	N/A
5233696	August 1993	Suzuki
N/A	**N/A**	N/A
5235686	August 1993	**Bosshart**
N/A	N/A	N/A
5295247	March 1994	Chang et al.
N/A	**N/A**	N/A
5299318	March 1994	**Bernard** et al.
N/A	N/A	N/A
5321830	June 1994	Nakamura et al.
N/A	**N/A**	N/A
5333292	July 1994	**Takemoto** et al.
N/A	N/A	N/A
5337415	August 1994	DeLano et al.
N/A	**N/A**	N/A
5355463	October 1994	**Moeller**
N/A	N/A	N/A
5357627	October 1994	Miyazawa et al.
N/A	**N/A**	N/A
5367571	November 1994	**Bowen** et al.
N/A	N/A	N/A
5371864	December 1994	Chuang
N/A	**N/A**	N/A
5379301	January 1995	**Sato** et al.

N/A	N/A	N/A
5394558	February 1995	Arakawa et al.
N/A	**N/A**	N/A
5394559	February 1995	**Hemmie** et al.
N/A	N/A	N/A
5425036	June 1995	Liu et al.
N/A	**N/A**	N/A
5430851	July 1995	**Hirata** et al.
N/A	N/A	N/A
5440632	August 1995	Bacon et al.
N/A	**N/A**	N/A
5442760	August 1995	**Rustad** et al.
N/A	N/A	N/A
5454100	September 1995	Sagane
N/A	**N/A**	N/A
5459844	October 1995	**Eickemeyer** et al.
N/A	N/A	N/A
5481713	January 1996	Wetmore et al.
N/A	**N/A**	N/A
5488710	January 1996	**Sato** et al.
N/A	N/A	N/A
5488729	January 1996	Vegesna et al.
N/A	**N/A**	N/A
5500942	March 1996	**Eickemeyer** et al.
N/A	N/A	N/A
5504923	April 1996	Ando
N/A	**N/A**	N/A
5509130	April 1996	**Trauben** et al.
N/A	N/A	N/A
5535329	July 1996	Hastings
N/A	**N/A**	N/A
5537629	July 1996	**Brown** et al.
N/A	N/A	N/A
5553301	September 1996	New et al.
N/A	**N/A**	N/A
5559975	September 1996	**Christie** et al
N/A	N/A	N/A
5560028	September 1996	Sachs et al.
N/A	**N/A**	N/A
5566298	October 1996	**Boggs** et al.
N/A	N/A	N/A
5600806	February 1997	Brown et al.
N/A	**N/A**	N/A
5619666	April 1997	**Coon** et al.
N/A	N/A	N/A
5623619	April 1997	Witt

N/A	**N/A**	N/A
5625787	April 1997	**Mahin** et al.
N/A	N/A	N/A
5630083	May 1997	Carbine et al.
N/A	**N/A**	N/A
5636374	June 1997	**Rodgers** et al.
N/A	N/A	N/A
5655097	August 1997	Witt et al.
N/A	**N/A**	N/A
5664134	September 1997	**Gallup** et al.
N/A	N/A	N/A
5689672	November 1997	Witt et al.
N/A	**N/A**	N/A
5694348	December 1997	**Guttag** et al.
N/A	N/A	N/A
5694587	December 1997	Webb et al.
N/A	**N/A**	N/A
5696955	December 1997	**Goddard** et al.
N/A	N/A	N/A
5713035	January 1998	Farrell, et al
N/A	**N/A**	N/A
5740413	April 1998	**Alpert** et al.
N/A	N/A	N/A
5742794	April 1998	Potter
N/A	**N/A**	N/A
5752259	May 1998	**Tran**
N/A	N/A	N/A
5764884	June 1998	Van Fleet
N/A	**N/A**	N/A
5790843	August 1998	**Borkenhagen** et al.
N/A	N/A	N/A
5790860	August 1998	Wetmore et al.
N/A	**N/A**	N/A
5794063	August 1998	**Favor**
N/A	N/A	N/A
5796972	August 1998	Johnson et al.
N/A	**N/A**	N/A
5796973	August 1998	**Witt** et al.
N/A	N/A	N/A
5796974	August 1998	Goddard et al.
712/211	**N/A**	N/A
5799144	August 1998	**Mio**
N/A	N/A	N/A
5829012	October 1998	Marlan et al.
711/102	**N/A**	N/A
5845102	December 1998	**Miller** et al.

N/A	N/A	N/A
5872946	February 1999	Narayan et al.
712/204	**N/A**	N/A
5884058	March 1999	**Narayan** et al.
N/A	N/A	N/A
5901225	May 1999	Ireton et al.
714/7	**N/A**	N/A
5920710	July 1999	**Tran** et al
N/A	N/A	N/A
5933629	August 1999	Mahalingaiah, et al
N/A	**N/A**	N/A
5983337	November 1999	**Mahalingaiah** et al.
712/32	N/A	N/A
6009513	December 1999	Mahalingaiah, et al.
N/A	**N/A**	N/A
6141740	October 2000	**Mahalingaiah** et al.
711/215	N/A	N/A
6158018	December 2000	Bernasconi et al.
714/8	**N/A**	N/A
6161172	December 2000	**Narayan** et al.
N/A	N/A	N/A
6192468	February 2001	Mahalingaiah, et al.
N/A	**N/A**	N/A

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY
US-CL		
259095	March 1988	EP
381471	August 1990	EP
03079031	April 1991	EP
640 916	March 1995	EP
0651320	May 1995	EP
2227584	August 1990	GB
459232	December 1991	GB
2263985	August 1993	GB
2263987	August 1993	GB
2281422	March 1995	GB

OTHER PUBLICATIONS

XP 000552115, "Breakpoints in Masked Microcontrollers To Fix Latent Software Defects", Michael Wilinson, Motorola Technical Developments, Nov. 1995, 1 page.

Mike Johnson, "Superscalar Microprocessor Design", AMD, Jan. 24, 1994, pp. 71-75.

Intel, "Pentuim Processor User's Manual", vol. 3, Architecture and Programming Manual, 1993, 7 pages.

Intel, "Pentium Processor Family User's Manual", vol. 1, Pentium Processor Family Data Book, 1994, 6 pages.

Michael Slater, "AMD's K5 Designed to Outrun Pentium", Microprocessor Report, Oct. 24, 1994, MicroDesign Resources, 7 pages.

Sebastian Rupley and John Clyman, "P6: The Next Step!", Pc Magazine, Sep. 12, 1995, 16 pages.

Tom R. Halfhill, "AMD K6 Takes On Intell P6", BYTE, Jan. 1996, 4 pages.

IBM Technical Disclosure Bulletin Entitled, "On-Site ROS Patch Mechanism", vol. 30, No. 5, Oct. 1987, New York, USA, pp. 158-160.

Scherpenberg, F.A., et al., Electronics De 1984 A 1985: Electronics Wee, "Asynchronous Circuits Accelerate Access to 256-K Read-Only Memory", pp. 141-145.

ART-UNIT: 2187

PRIMARY-EXAMINER: Gossage; Glenn

ATTY-AGENT-FIRM: Conley, Rose & Tayon, PC Kowert; Robert C.

ABSTRACT:

Random access memory (RAM) may be provided in a processor for implementing microcode patches. The **patch** RAM may loaded by a microcode routine that is part of the normal microcode contained in a microcode read only memory (**ROM**) unit of the processor. When the processor powers-up, it uses its internal **ROM** microcode only if no **patches** are installed. If patches are installed and a microcode line is accessed for which a patch is enabled, the patch is executed instead of the microcode line. A **patch** may be enabled by setting a match register with the address of the microcode instruction line in the microcode **ROM that is to be patched**. Whenever the microcode **ROM** address matches the contents of a match register, control is transferred to the **patch** RAM. The patch RAM may have a plurality of fixed entry points each corresponding to a different match register.

38 Claims, 6 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 6

---------- KWIC ---------

Abstract Text - ABTX (1):

Random access memory (RAM) may be provided in a processor for implementing microcode patches. The **patch** RAM may loaded by a microcode routine that is part of the normal microcode contained in a microcode read only memory (**ROM**) unit of the processor. When the processor powers-up, it uses its internal **ROM** microcode only if no **patches** are installed. If patches are

installed and a microcode line is accessed for which a patch is enabled, the patch is executed instead of the microcode line. A **patch** may be enabled by setting a match register with the address of the microcode instruction line in the microcode **ROM that is to be patched**. Whenever the microcode **ROM** address matches the contents of a match register, control is transferred to the **patch** RAM. The patch RAM may have a plurality of fixed entry points each corresponding to a different match register.

Brief Summary Text - BSTX (17):

Another problem with fetching microcode patches from external memory, or even from internal caches, is performance. In many conventional processors, the width of data returned by memory or cache accesses is smaller than the width of microcode instructions fetched from the microcode ROM of the processor. Thus, if a microcode **patch** is fetched from external memory or from a cache, multiple memory accesses will be required to load a **patched** microcode instruction, as compared to a single wide fetch from the processor's microcode **ROM**. Furthermore, the latency for memory accesses is typically much longer than for fetches from the internal microcode ROM. Thus, microcode patches fetched from external memory or cache typically have an adverse effect on processor performance since fetching such a patch typically requires more and slower accesses.

Brief Summary Text - BSTX (19):

Another problem with conventional microcode patch mechanisms concerns triggering the patch. One technique has been to provide a tag memory in the processor having one bit for every location in the microcode ROM. If a particular microcode **ROM** location is to be **patched,** then the corresponding bit is set in the tag memory. However, for typical microcode ROM sizes, this technique may require thousands of bits of tag memory. Additionally, timing may be complicated to access all the bits of the tag memory for each microcode **ROM** fetch in order to check if a **patch** is enabled.

Brief Summary Text - BSTX (22):
   An amount of random access memory (RAM) may be provided in a processor for implementing microcode patches. The **patch** RAM may loaded by a microcode routine that is part of the normal microcode contained in a microcode **ROM** unit of the processor. When the processor powers-up it uses its internal **ROM** microcode only if no **patches** are installed. However, if patches are installed and if a microcode line is accessed for which a patch is enabled, the patch is executed instead of the microcode line.

Brief Summary Text - BSTX (23):
   A **patch** may be enabled by setting a match register with the address of the microcode instruction line in the microcode **ROM that is to be patched**. A processor may include several such match registers. Whenever the microcode **ROM** address matches the contents of one of the match registers, control is transferred to the **patch** RAM. The patch RAM may have a plurality of fixed entry points each corresponding to one of the match registers. Thus, when an MROM address matches a match register, control is passed to the patch RAM at the fixed entry point corresponding to the matching match register. To disable a match register, its contents may be written with a value

that will never match any ROM address, e.g. -1.

Brief Summary Text - BSTX (24):

Whenever a match is detected between an MROM address and a match register, the microcode instruction line from the **ROM** is disabled and control is transferred to the appropriate entry point in the **patch** RAM. In some embodiments, a delay slot may also be issued from the **ROM** while control is being transferred to the fixed entry point in the **patch ROM**. Thus, there may be a two cycle bubble in the MROM unit pipeline whenever control is transferred from the microcode **ROM to the patch** RAM since both the matching address line and the delay slot line from the **ROM** are cancelled. In a preferred embodiment, the **patch** RAM is a contiguously addressed extension of the microcode **ROM**. Therefore, regular microcode jump or branch instructions may be used when exiting a **patch** routine to return to the **ROM**. Thus, when exiting a patch routine there is no need to cancel any instructions and patch routines may be exited and MROM operation resumed with no delay.

Detailed Description Text - DETX (33):

Turning now to FIG. 3, a more detailed illustration is provided of a portion of MROM unit 34 that illustrates a mechanism for implementing patches to the microcode. MROM storage 64 may include a read only memory (**ROM**) portion 64a and a **patch** RAM portion 64b. The ROM portion 64a is where microcode instruction lines are typically accessed. In a preferred embodiment, **ROM** 64a holds up to 3K microcode lines or triads and **patch** RAM 64b stores up to 64 triads. In a preferred embodiment, **patch** RAM 64b exists in the same address space as **ROM** 64a and is addressed contiguously with **ROM** 64a. For example, in the embodiment in which the size of **ROM** 64a is 3072 (3K) lines and the size of **patch** RAM 64b is 64 lines, the microcode address range for **ROM** 64a would be 0x000 to 0xBFF and the address range for **patch** RAM 64b would be 0xC00 to 0xC3F. A line from **ROM** 64a or **patch** RAM 64b is selected according to an address from next address register 94. The selected line is provided to output register 80. In a preferred embodiment, each line or triad includes 3 microcode instructions OP1, OP2 and OP3, and a sequence control field. The sequence control field may include a branch target address and a control portion. The control portion may include information to determine whether or not the branch should be taken. The microcode instructions OP1 through OP3 are output to MROM early decode 66.

Detailed Description Text - DETX (35):

The address from next address register 94 is also supplied to a comparator 84. Comparator 84 compares the address to values stored in a number of match registers 88. In one embodiment, match registers 88 include eight match registers where each match register stores 12 bits corresponding to a location in the 12 bit address space of MROM storage 64. Values are programmed into the match registers as described below. If the address from next address register 94 matches one of match registers 88, then a patch from patch RAM 64b will be implemented. When an address from next address register 94 matches the address stored in one of match registers 88, comparator 84 selects a corresponding patch **RAM** address from **look up table** 90. The selected address in look up table 90 is supplied to MUX 86. In a preferred embodiment, the addresses in look up

table 90 are hard wired to correspond to particular ones of match registers 88.
An example of look up table values for particular match registers for one embodiment is as follows:

Detailed Description Text - DETX (37):

Thus, when the next MROM storage address matches one of match registers 88, the MROM access sequence jumps to the address indicated by look up table 90. The microcode line that was fetched from **ROM** 64a into output register 80 is cancelled and a **patch from patch RAM** 64b is executed instead starting from the address indicated by **look up table** 90. Any delayed branch effects from jumping to the patch RAM are also cancelled. For example, in embodiments where branches are delayed by one cycle, both the line from **ROM** 64a that triggered the **patch** and the next line (the branch delay slot) are cancelled. Thus, in such an embodiment, whenever MROM unit 34 switches to a microcode patch in patch RAM 64b a two-cycle bubble in the MROM pipeline will be incurred.

Detailed Description Text - DETX (39):

In a preferred embodiment, the patch **RAM** locations indicated by **look up table** 90 form a vector table that points to a location in the rest of patch **RAM** 64b where the rest of the patch routine is located. In the example described above for look up table 90, the vector table entry points are located at every other address (e.g. offsets 00, 02, 04. . .) to allow for branch delay slots at the intervening addresses (e.g. offsets 01, 03, 05. . .)

Detailed Description Text - DETX (40):

Once control is transferred to **patch** RAM 64b execution continues from **patch**

RAM 64b until a **patch** line jumps back into the **ROM**. In a preferred embodiment, in which the **patch** RAM exists as part of the MROM address space, a jump from the **patch** RAM 64b to **ROM** 64a may be treated as any other microcode jump or branch. Thus, while there may be a two cycle bubble when entering the **patch** RAM 64b as described above, there is no delay when jumping back to **ROM** 64a from **patch** RAM 64b. There is no delay when jumping back to **ROM** 64a because there is no need to cancel a **patched** microcode line or branch delay slot. The **patch** routine simply includes a normal microcode jump with a **ROM** target address.

Detailed Description Text - DETX (41):
   Thus, using the mechanism described above in FIG. 3, any microcode line of **ROM** 64a may be **patched** by a routine loaded in **patch** RAM 64b. To invoke a patch, one of match registers 88 is programmed with the address corresponding to the line of MROM 64a desired to be patched. Note that the sizes and address ranges of **ROM** 64a and **patch** RAM 64b described above were merely examples for certain embodiments, other sizes may be employed as desired. Similarly, the number of match registers may be varied. Additionally, instead of being hard wired, in some embodiments look up table 90 may be programmable. Also note that sequencer 92, MUX 86, incrementer 82, look up table 90, match registers 88 and comparator 84 may all be considered to be part of sequence controller 65 from FIG. 2.

Detailed Description Text - DETX (48):
   The header may also include a flag to indicate that a particular patch should be executed immediately after the patch RAM is loaded. In the example

above, an init flag is included for this purpose. When this flag is set it causes a jump to a particular patch RAM location immediately after patch RAM 64b is loaded. This feature allows the patch RAM to contain a microcode routine that is run immediately after the patch is loaded and before normal processor operation resumes. Such a **patch** routine that is executed immediately after loading the **patch** RAM may be useful for fixing or changing internal processor values that are not associated with microcode **ROM** routines and therefore cannot be **patched** by setting one of match registers 88 to correspond to a line in **ROM** 64a. For example, it may be necessary to change an internal processor configuration register to disable a hardware optimization that has been determined to be faulty. A microcode patch may be loaded into patch RAM 64b to appropriately change the internal configuration register. That patch may be placed at the init entry point which is jumped to after the patch RAM is loaded when the init flag is set. Typically this type of patch only needs to be run once, such as during a power-up software routine. When such a patch is loaded and the init flag is set, the microcode patch RAM loader will notice that the init flag is set and jump to the init entry point and the patch will be executed. Note that this mechanism provides a second way to enter patch RAM 64b in addition to using match registers 88. In a preferred embodiment, the init entry point in patch RAM 64b is a fixed location. This mechanism may be used to fix other processor bugs in addition to bugs in the microcode itself.

Detailed Description Text - DETX (50):

The header also includes values for the match registers 88. The match register values may indicate an address value for each match register which is to trigger a patch. The address value to be loaded in a particular match register corresponds to the address of a line or triad in ROM 64a. As described, if **ROM** 64a is accessed at an address matching an address loaded in one of the match registers, then a **patch** is executed from **patch** RAM 64b instead of the accessed **ROM** line and/or delayed branch slot from the **ROM**. In a preferred embodiment, each match register indicates a fixed entry point into patch RAM 64b as described above. A match register and its corresponding patch may be disabled by setting the value in the microcode patch block header for that match register with a value that will not match any MROM address. In a preferred embodiment, a match register is disabled by setting it to -1 (e.g. 0xFFF).

Detailed Description Text - DETX (55):

Turning now to FIG. 4, a diagram is provided illustrating a method of operation for microcode patching. One or more patches are loaded into a patch RAM, such as patch RAM 64b of FIG. 3, as indicated at 100. In one embodiment, the one or more microcode patches may be stored in the system memory of a computer system. A microcode **patch** loader stored in microcode **ROM** 64a (FIG. 3) may be called to load the microcode **patch** data from the system memory into the microcode **patch** RAM. Also, as indicated at 100, values are set for match registers. These values may also be read from system memory, such as in the microcode patch header described above. An MROM routine may begin (102). During the execution of MROM routines, MROM addresses are generated in the microcode unit to access microcode instruction lines in the microcode memory

(104). As each MROM address is generated, the address is compared to the match registers, as indicated at 106. If no match is detected (at 108), the addressed microcode line is dispatched from the microcode memory (e.g. ROM 64a) to be decoded and executed and normal operation continues, as indicated at 118. However, if a match is detected (at 108), the microcode unit jumps to a patch RAM location corresponding to the matching match register, as indicated at 110. Any MROM line and delayed branch slots that were dispatched from the microcode ROM when the address that triggered the match was generated are cancelled, as indicated at 112. Thus, instead of executing the microcode instruction line indicated by the address that matched one of the match registers, a patch is executed from the patch RAM, as indicated at 114. Upon the completion of the **patch, the patch** routine jumps back to the microcode **ROM,** as indicated at 116. Typically the **patch** will jump back to the next **ROM** address that would have been accessed from the microcode **ROM** before the **patch** was initiated. Note that the diagram of FIG. 4 is merely illustrative of the logical operation of a microcode patching method according to one embodiment. FIG. 4 is not meant to necessarily imply a specific pipeline sequence of operations.

Detailed Description Text - DETX (56):

Turning now to FIG. 5, a diagram is provided illustrating a method for loading microcode patches. First, any desired microcode patches are written and assembled (into a microcode patch block (MPB) for example) as indicated at 120. Typically, such patches are written and assembled by a processor manufacturer. However, in some embodiments users may be able to write and assemble their own patches. Typically a microcode patch is written and assembled like other microcode routines. However, the microcode patches may be formatted in a particular format, such as described above in which the patches are located in a microcode patch block having a header and a patch data portion. Typically system software, such as an operating system or BIOS, stores the microcode patch block in system memory, as indicated at 122. In some embodiments, the patch block may be made accessible only in certain processor mode(s) and/or current privilege level(s) (CPL). The microcode patches may then be loaded into the patch RAM. This may be accomplished by calling a microcode patch RAM loader, as indicated at 124. In a preferred embodiment, the **patch** RAM loader is a microcode routine stored in the microcode **ROM**. In a preferred embodiment, the patch RAM loader function is called via a write to a model specific register with another register pointing to the microcode patch block in system memory. The MSR used to invoke the patch RAM loader (PRL) may be referred to as MSR PRL, and the register used to point to the location of the patch block in memory may be referred to as the pointer register. Thus, to load a microcode patch block, the linear address of the start of the microcode patch block in system memory is loaded in the pointer register and a write model specific register (WRMSR) instruction to MSR PRL is executed, as also indicated at 124. The patch RAM loader then installs the patch data as indicated at 126. In one embodiment, paging and segmentation are arranged such that the patch RAM loader does not encounter any page faults or segmentation faults while accessing the microcode patch block in system memory. This allows the microcode patch loader routine to be simplified. If the patch data is successfully installed, a processor register may be updated with a patch ID from the microcode patch block header to indicate the successful installation, as also indicated at 126. In a preferred embodiment, this register is MSR 8Bh. The microcode patch RAM loader then checks to see if the init flag from the microcode patch block header is set, as indicated at 128. If the flag is not set, then the patch installation is complete and normal processor operation resumes, as indicate at 132. If the init flag is set, then an init patch is executed, as indicated at 130. For example, if the patch RAM loader determines that the init flag is set, it jumps to a fixed location, e.g. 0xC10, which locates the beginning of the initialization patch. As mentioned above, executing such an initialization **patch** may be desirable if it is necessary to fix or change something in the processor that is not associated with a **ROM** microcode routine, and therefore a normal **patch** using the match registers cannot be triggered. For example, if one of the internal configuration registers of the processor needs to be changed in order to disable a faulty hardware optimization, an initialization patch may be executed to implement that change.

Detailed Description Text - DETX (57):

Typically the loading of the patch data into the patch RAM is performed by the system software or BIOS during initialization of the processor, e.g. after power-up or reset. However, in some embodiments, a software application may load a **patch into the patch** RAM and use the **patch** to redefine certain lines of microcode instructions in the microcode **ROM**. For example, an application could use a patch to redefine a microcoded instruction by setting a match register to the address that indicates the beginning of the microcode routine to implement

the microcoded instruction that is intended to be redefined. The application
can then provide its own microcode as a patch to change the definition of the
instruction. However, in typical embodiments, the microcode patch block ID's
are not made available to application users and the microcode patch block data
is encrypted so that the use of microcode patches may be controlled.

Current US Cross Reference Classification - CCXR (1):
711/102

Current US Cross Reference Classification - CCXR (3):
711/202

```
US-PAT-NO:                5790856

DOCUMENT-IDENTIFIER:   US 5790856 A

TITLE:                    Methods, apparatus, and data
structures for data driven
                         computer patches and static analysis of
same

DATE-ISSUED:              August 4, 1998

INVENTOR-INFORMATION:
NAME                               CITY
STATE     ZIP CODE  COUNTRY
Lillich; Alan W.                   Los Gatos                CA
     N/A       N/A

ASSIGNEE INFORMATION:
NAME                          CITY                          STATE
ZIP CODE  COUNTRY   TYPE CODE
Apple Computer, Inc.          Cupertino                     CA
N/A       N/A       02

APPL-NO:        08/ 436945

DATE FILED:     May 8, 1995


INT-CL:              [06] G06F009/45

US-CL-ISSUED:        395/703, 395/712

US-CL-CURRENT:       717/163, 717/168


FIELD-OF-SEARCH:  395/700; 395/650 ; 395/705 ; 395/710 ;
395/712 ; 395/653
                  ; 395/703


REF-CITED:
                             U.S. PATENT DOCUMENTS
    PAT-NO                ISSUE-DATE            PATENTEE-NAME
                     US-CL

 5155847           October 1992       Kirouac et al.
```

395/600	**N/A**	N/A
5325533	June 1994	**McInerney** et al.
395/700	N/A	N/A
5359730	October 1994	Marron
395/650	**N/A**	N/A
5369766	November 1994	**Nakano** et al.
395/700	N/A	N/A
5369770	November 1994	Thomason et al.
395/725	**N/A**	N/A
5375241	December 1994	**Walsh**
395/700	N/A	N/A
5408665	April 1995	Fitzgerald
395/700	**N/A**	N/A
5481713	January 1996	**Wetmore** et al.
395/705	N/A	N/A
5519866	May 1996	Lawrence et al.
395/700	**N/A**	N/A

OTHER PUBLICATIONS

Thompson et al., "Apple, IBM Bring Power To the Desktop," Byte Magazine,
Apr. 1994, p. 44, vol. 19, No. 4.

Thelen, Randy, "Under the Hood: the Power Mac's Run-Time Architecture,"
Byte Magazine, Apr. 1994, p. 131, vol. 19, No. 4.

Apple Computer Inc., "Insided Macintosh: PowerPC System Software,"
Addison-Wesley Publishing Co, pp. 1-20--1-34 and pp. 1-66--1-68, 1994.

"Inside Macintosh: PowerPC System Software," Apple Computer, Inc., 1994,
Addison-Wesley Publishing Company.

ART-UNIT: 236

PRIMARY-EXAMINER: Kriess; Kevin A.

ASSISTANT-EXAMINER: Corcoran, III; Peter J.

ATTY-AGENT-FIRM: Hickman Beyer & Weaver,LLP

ABSTRACT:

The present invention teaches a variety of methods, apparatus and data structures for providing data driven patching. According to one embodiment, patches are stored in a known format in a discernible location. In the described embodiment, each fragment code may have a corresponding patch library. This enables the patches to be located and analyzed in a quiescent state. In a method aspect of the present invention, the operating system, or a separate utility program, can evaluate and selectively add patches. Therefore, the present invention introduces a patch integrity validation layer into the patching process. In another method aspect, the invention teaches evaluating the patches in a quiescent state whereby the patches introduced by a program or a combination of programs may be exhaustively evaluated prior to execution.

41 Claims, 14 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 10

---------- KWIC ---------

Brief Summary Text - BSTX (9):

The system routines for the 68K operating system reside mainly in ROM. However, to provide flexibility for any subsequent development, application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM. This indirect mechanism permits the **ROM** addressing of system routines to vary, or to be replaced by **patch** routines, without affecting the operation of applications which utilize the system routines.

Brief Summary Text - BSTX (10):
The 68K **patching** paradigm 100 includes application code 102 having at least
one ATRAP instruction 104, low memory locations 106, a trap dispatcher 108, a
trap table 110, **ROM** 120 having system code 122, and RAM 130 including at least
one **patch** code 132. While the operating system routines reside mainly in ROM
120 (in their original state), information regarding the locations of the
operating system routines is encoded in compressed form within ROM 120. Upon
system start up, this information is decompressed and the trap table 110 is
formed in RAM 130.

Brief Summary Text - BSTX (16):
One example of a redirection to a patched system routine is symbolized in
FIG. 2 by dashed flow control lines 150 and 152. Similar to the non-patched
system routine described previously, an ATRAP instruction 104 **calling** a patched
system routine will initiate a process in which the trap dispatcher 108 will
look up the system routine corresponding to the ATRAP instruction 104.
However, in the patched case, address.sub.-- 1 will point to patch code 132
located in RAM 130 rather than the original system routine. Thus dashed flow
control line 150 illustrates jumping to the patch code 132 and dashed flow
control line 152 illustrates jumping back to the application code 102 after the
patch code 132 has finished executing.

Current US Cross Reference Classification - CCXR (1):
717/168